TDD Exercise: Tax Calculator

In this exercise, you will help the Swedish tax agency (Skatteverket) to compute taxes. The actual calculation is quite complicated, so we have reduced scope to only take into account the base income and place of residence, the municipality.

Your goal is to test-drive a computation engine that will accept two parameters: base income (a number) and municipality name (a string) and compute the final tax based on the business rules below.

Depending on your experience with TDD, you can pick one of the following approaches:

Test-drive a class/module

If you're just trying out TDD, aim for test-driving a class/module that performs the computation, for example something like:

```
double finalTax = TaxCalculator.ComputeTax(300000, "Stockholm");
```

Test-drive an interactive application

Make the engine interactive somehow: console, web application, or window application. You can get some inspiration by looking at `SimpleOsCommandExecutor.cs`.
A console application may operate like this:

```
EdeklarationConsole.exe 300000 Stockholm
```

Test-drive an interactive application using the London school

Make the engine interactive somehow: console, web application, or window application, and explicitly use the London school approach. Start with an end-to-end test and work from the user interface inwards stubbing and mocking collaborators as you approach the fringe classes. If you're using C# and aiming for a console application, use `TaxCalculatorEndToEndTest` to get started.

Strict TDD using the TPP

Pick your interface and entry point, and try test-driving the engine using the Transformation Priority Premise (https://8thlight.com/blog/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html). Since you'll be spending some time reading up on this, you most likely won't be able to write more than a handful of tests. Be prepared to share your experiences with the class tough.

Business Rules

1. Every tax payer is entitled to a standard deduction of 13 200 SEK. Incomes below this threshold yield no tax.
2. Every tax payer pays a municipality tax, which is dependent on the place of residence. Some municipalities and their tax rates are presented in the table below.
3. Income bracket 1 is at 438 900 SEK. After that, a tax payer starts paying a 20% "state tax." However, because of the standard deduction, the state tax is effectively paid above 452 100 (438 900 + 13 200) SEK.
4. Income bracket 2 is at 638 500 SEK, where an additional 5% is added to the state tax. Again, because of the standard deduction, the higher tax is effectively paid on incomes greater than 651 700 (638 500 + 13200) SEK.

Since the point of the exercise isn't to make educated guesses about how to interpret the business rules, there are plenty of examples with real numbers in `Example.pdf`.

Municipalities and their tax rates

Municipality	Tax rate
Botkyrka	32.47
Haninge	32.00
Stockholm	30.05
Nacka	30.85
Sundbyberg	31.22
Vaxholm	32.10
Tierp	32.94
Uppsala	33.09
Flen	33.34
Habo	33.67

Tips and suggestions

- ✓ Don't forget to slice your functionality and deliver iteratively and in increments. It's more rewarding to produce a solution that works all the way for one particular case, than having something that covers all business rules, but doesn't compile.
- ✓ Don't get stuck in input/output. You need to present the result of the calculation, but leave error handling for last.
- ✓ Don't forget the Single Responsibility Principle, especially if you do the London school approach.

Tips and suggestions for the mockist approach

The problem is algorithmic in its nature, and the class(es) that do the calculation are “fringe” classes that will require plain asserts. There’s room for some collaborating objects though:

- ✓ Something needs to read the user’s input
- ✓ Something performs the calculation
- ✓ Something knows about the different municipalities and their tax rates

The challenge is to figure out how many classes/modules the above will require and how they communicate.